Sequential Models for Graphs

Ricardo Buitrago
ricardob@cs.cmu.edu

Richa Gadgil
rgadgil@cs.cmu.edu

Bao Nguyen
manhbaon@cs.cmu.edu

1 Motivation

Graphs are a natural structure to represent and reason over complex interactions between individual entities, such as in social networks or molecular graphs. Most attempts to design Deep Learning architectures to learn representations over graphs initially relied on message passing neural networks (MPNNs) (Gilmer et al. [2017]). These models iteratively update the representation of a target node by aggregating the representations of its neighbors, which leverages the structure of the graph but fails to account for long-range interactions (LRI). Subsequently, Graph Transformers (Dwivedi and Bresson [2021]) have given better results on the classical benchmarks, since all the nodes directly interact with each other to produce their representations. However, their quadratic complexity is a bottleneck, and the structure of the graph is not inherently leveraged. Recurrent architectures have witnessed a resurgence with state-space models (SSM) (Gu et al. [2022]), which have had promising results on efficient sequence modeling, and thus have prompted initiatives to adapt them to graph tasks. Recent works use Mamba (Gu and Dao [2023]) to conduct node prioritization for more efficient context-aware reasoning or adapt message passing on learned representations. The recent papers trying to leverage SSMs in different ways to capture graphical interactions claim to obtain state-of-the art results, but it is unclear what is the soundest way to integrate and compare the latest recurrent modules in the MPNNs/GT paradigm. Hence our motivation to quantify the potential of the latest SSM Mamba as an alternative to traditional Message Passing, through the empirical performance of a newly proposed architecture and its relation to the classical bottleneck of oversmoothing. Our code is publicly available at https://github.com/r-buitrago/SeqForGraphs.

2 Background

For our mid-term report, we surveyed the literature on neural architectures for inference on graphs, which we recall and extend in the next section 3. We figured out that the most modular architecture was to be taken after GraphGPS Rampášek et al. [2023], where local and global interactions are leveraged. We studied how to use sequential models for these local and global interactions. The global interactions are proposed in Wang et al. [2024], where the whole graph is serialized into a sequence to be fed to Mamba after message passing. For the local interactions part, GRED Ding did one serialization of the graph per node, based on the node's neighbourhood. In the midway report, we had preliminary results in which the baseline (a simple message passing neural network) outperformed both GRED-based models and global-interaction models. Nevertheless, we have found several training and architecture details that change the results, as we explain in this final report.

3 Related Work

3.1 Architectures for Graph Representation Learning

Message Passing Neural Networks (MPNN): Message-passing neural networks are a class of GNNs that iteratively aggregate local neighborhood information to learn the node/edge representations (Kipf and Welling [2017]). The most popular variants include GCN (Henaff et al. [2015]), GatedCNN (Bresson and Laurent [2018]). Popular because they intuitively leverage the graph structure, they still

Preprint. Under review.

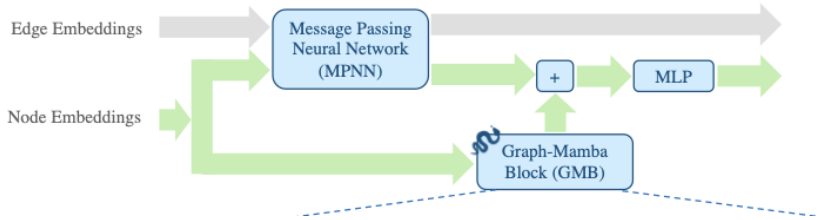


Figure 1: From Wang et al. [2024], Block of joint representation update, locally (e.g. MPNN here) and globally (e.g. GMB here, or regular transformer in Rampášek et al. [2023]). We will not be using it beyond the background-preliminary work for its lack of theoretical and empirical groundness so far - as we focus on Mamba for Message Passing.

have an expressivity limited to the 1-WL isomorphism test (Xu et al. [2019]), and are prone to the issues of oversquashing(Topping et al. [2022]) and oversmoothing (Rusch et al. [2023]), which we detail below. That is why the community has tried to improve MPNNs baseline results on Long-Range Dependency benchmarks (Dwivedi et al. [2023]). Different MPPNs variants, like SPN (Abboud et al. [2023]) empirically alleviate over-squashing, but the way they model the interactions from distant nodes often hampers the expressivity of the network. As of April 2024, a very recent work by Errica et al. [2024] introduces Adaptive Message Passing, where additional parametrization allow message and depth selection along the message passing hops. This allegedly yields unprecedented improvements on benchmarks by directly tackling underreaching (another qualitative challenge we detail below), but we will not analyze it in this work since we focus on the specific candidate alternative of State-Space Models to address similar bottlenecks.

Graph Transformers (GT): Graph Transformers tackle some shortcomings of MPNNs since every node can directly attend to all the others, allowing a direct flow of information. However, despite attempts to add positional encodings (PE) to account for some graph structure (e.g. with the graph Laplacian, Dwivedi et al. [2022]), the order of the nodes in the graph is often brushed over. To alleviate the quadratic complexity of the attention computation, some models try to approximate it with sparse attention, like Performer (Choromanski et al. [2022]) - which was adapted for graphs in Exphormer(Shirzad et al. [2023]). Recent approaches try to come back to simpler, more efficient and more intuitive inductive biases, far from the quadratic complexity to compute positional encoding and subsequent inference with Transformers.

Sequential Models for Graphs: With a more efficient inference and a sequential inductive bias, State-Space Models (SSM) use a recurrent scan over the input, store context in their hidden states, and update the output with new inputs. The first modern SSM, S4, was introduced recently by Gu et al. [2022], and many variants been applied to vision and audio modalities. However, on most difficult benchmarks, transformers were more expensive but still prevalent, until Mamba (Gu and Dao [2023]) introduced a data-dependent selection mechanism to capture long-range context with increasing sequence length. The promises of these SSMs (S4, Mamba) have revived an interest in recurrent architectures in general. For instance, LRU (Orvieto et al. [2023]) lies somewhere between RNNs and SMMs, being directly designed for a discrete-time system (unlike SSMs) but endowed with a complex parametrization for stable signal propagation (unlike RNNs). If Mamba was successfully adapted for non-sequential inputs like images, there is no clear best adaptation of it for graph-related tasks, due to their intricate structure. So far, we have compiled four recent preprints trying to incorporate SSMs in the graph formalism, using S4 (Song et al. [2024]), LRU (Orvieto et al. [2023]), or Mamba (Wang et al. [2024], Behrouz and Hashemi [2024]). Unpublished, these drafts constitute for us references that confort the potential of Mamba to model graphical long-range dependencies. However, these recent works only mostly focus on deriving complicated high-level architecture adaptations suitable for graphs: this is a prerequisite, but fails to justify theoretically or empirically that introducing a SSM alleviates classical long-range modeling pitfalls.

3.2 Challenges and benchmarks for Graph Representation Learning

Oversmoothing: Node features in GNNs tend to become more similar with the increase of the network depth, a recurrent effect empirically observed and coined "oversmoothing". This is counterintuitive since the more layers node embeddings have to traverse to contribute to the representation of another

target node, the more expressive this representation is. This effect has been extensively studied Cai and Wang [2020], and some metrics have been introduced to measure oversmoothing, including *Dirichlet Energy* and *Mean Average Distance* (MAD). In the idea, they both evaluate the similarity between the embedding of a given node and the embeddings of its neighbors - and average this across all nodes. Rusch et al. [2023] introduce a formal definition of node-similarity measure and points out that only the Dirichlet Energy (which uses the L^p distance between embeddings, with p = 2 canonically) is a proper similarity metric, while MAD (which uses the scalar product between these embeddings) is always 0 for scalar features. We will thus use the former for our experiments, which is elaborated on in section 4. Rusch et al. [2023] also evidences that trying to keep a constant Dirichlet Energy (i.e. difference between adjacent node embeddings between layers) is a very hard constraint for the expressivity of the GNN, thus being a compromise. Some methods to alleviate this tendency include normalizing the features, introducing residual connections, etc. which are out of scope for our study, since we are verifying the intrinsic properties of SSMs to mitigate this.

4 Methods

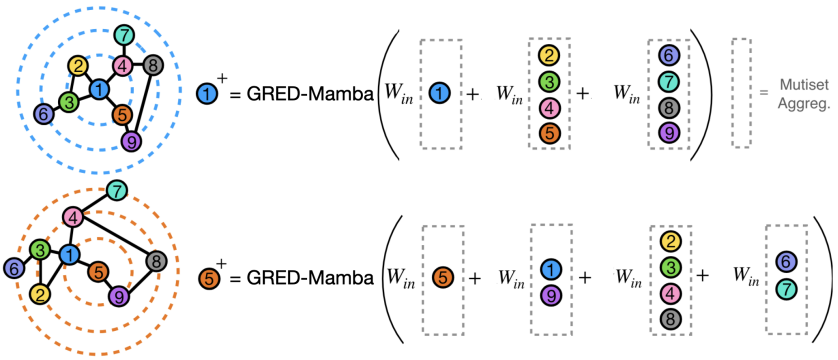


Figure 2: Depiction of the GREDMamba layer operation for two different target nodes. The gray rectangular boxes indicate the application of multiset aggregation. Image modified from 4.1

4.1 GRED Message Passing Algorithm

In this section we explain **GRED** (Ding et al. [2024]), which is a Message Passing Neural Network that uses a sequential model (as opposed to GineConv Hu et al. [2020], which generally uses Linear layers). This Message Passing works in the following way:

1. For each root node in the graph, a sequence is built by aggregating all the neighbouring nodes in the graph in the following way: the i-th element of the sequence is the mean of the features of the nodes whose distance to the root is i. This is done up to distance K (this neighbourhood is called the K-hop neighbourhood). From now on, every time we mention K or K-hop neighbourhood, we will refer to this parameter.
2. After the neighbourhood is arranged into a sequence, a sequential model is applied in reverse order (ie, starting from the nodes that are furthest away and ending in the root).
3. The features of the root are updated with the final output of the sequence model. In essence, the root is updated based on a compressed representation (memory state) of its neighbouring nodes.

In its original implementation, Ding et al. [2024] uses LRU as the sequence model, but we will use Mamba as our sequence model. We will also do an ablation on the influence of the graph processed as a sequence and the effect of Mamba on graphs with long-range dependencies.

4.2 Models

In this report, we propose two new models inspired in the latest literature and will compare their performance against the baseline. The three models we will consider will be:

- **GINEConv (Baseline):** It is a simple GINEConv MPNN Hu et al. [2020], without the Post Message Passing Sequence Model. For the Neural Network, we choose a two layer fully connected MLP with ReLU activation between the layers.

- **GREDMamba (Ours):** This models uses a GRED-style MPNN, which we explained before in the Local Interactions. We extend the original GRED architecture and use Mamba as the Sequence Model. It has no Global Sequence Model.

  We propose this architecture for two reasons. First, Mamba is specifically designed with discrete data, which is an advantage over other SSM-based models. Second, Mamba has the capacity to selectively forget or hold the memory based on the input. We hypothesize that this will be very useful with graphs, given that when doing message passing, the information of nodes that are distant normally gets lost when reaching the destination node. To the contrary, Mamba is capable of holding memory from far away nodes, and thus this kind of oversquashing is limited.

- **GREDMamba-Mamba (Ours):** This model is based on both the GRED method and the very recent Graph-Mamba preprint (Wang et al. [2024]). It uses GRED with Mamba during MPNN (Local interactions), and after, it applies Mamba again in the Global interactions stage. As we already mentioned, in contrast to the Local Interactions, which serialize the graph into a sequence for every node and then apply Mamba to it, in the Global Sequence step the graph is serialized only once. Following (Wang et al. [2024]), this global serialization is done by sorting the nodes by their degree. We introduce this new architecture, which we did not have in the midway report, because we believe that it can help model global interactions.

4.3 GREDMamba and GREDMamba-Mamba layers architecture

In the Midway report, our results were significantly worse than the baseline (GineConv). Initially we had the problem that our results diverged when the number of layers or the K parameter were large. After doing an architecture search, we found two configurations that worked best, explained in 1 and 2. Although there are many details in the layers, there are two keys. First, layer norms must be applied at the beginning of each layer (we found that prenorm stabilized training), and they must also be applied right before the sequential model. Secondly, two residual connections are used, one applied to the serialized features before the sequential model, and another one after the sequential model. MLP layers allow mode mixing, which we observed to help performance.

The full models consist of an node embedding encoder, several GREDMamba or GREDMamba-Mamba layers, a final *global add pool* and a final MLP to regress the desired variables of the graph.

Algorithm 1 GREDMamba Layer Operation

Inputs: Graph $\mathcal{G} = (U, V)$; $\{x_u^{(l)} | u \in U\}$ (node features of previous layer); $\{\text{GRED}_u | u \in U\}$ (serialization methods for each node, explained in 4.1).
Output: $\{x_u^{(l+1)} | u \in U\}$ (node features after layer)
for all $u \in U$ **do**
    $\left(s_{u,i}^{(l)}\right)_{i=1}^{K} \leftarrow \text{GRED}_u \left[\{\text{LayerNorm}\left(x_w^{(l)}\right) | w \in U\}\right]$
    for $i = 1$ to K **do**
        $s_{u,i}^{(l)} \leftarrow \text{LayerNorm}\left(\text{MLP}\left(s_{u,i}^{(l)}\right) + s_{u,i}^{(l)}\right)$
    end for
    $h^{(l)} \leftarrow x_u^{(l)} + \text{Mamba}\left[\left(s_{u,i}^{(l)}\right)_{i=1}^{K}\right][K]$
    $x_u^{(l+1)} \leftarrow h^{(l)} + \text{MLP}\left(h^{(l)}\right)$
end for
Note: [K] denotes indexing by element K.

Algorithm 2 GREDMamba-Mamba Layer Operation

1: **Inputs:** Graph $\mathcal{G} = (U, V)$; $\{x_u^{(l)} | u \in U\}$ (node features of previous layer); $\{\text{GRED}_u | u \in U\}$ (serialization methods for each node, explained in 4.1); GS (global serialization, in our case ordering by degree).
2: **Output:** $\{x_u^{(l+1)} | u \in U\}$ (node features after layer)
3: **for all** $u \in U$ **do**
4: $\left(s_{u,i}^{(l)}\right)_{i=1}^{K} \leftarrow \text{GRED}_u \left[\{\text{LayerNorm}\left(x_w^{(l)}\right) | w \in U\}\right]$
5: **for** $i = 1$ to K **do**
6: $s_{u,i}^{(l)} \leftarrow \text{LayerNorm}\left(\text{MLP}\left(s_{u,i}^{(l)}\right) + s_{u,i}^{(l)}\right)$
7: **end for**
8: $h_{\text{local}}^{(l)} \leftarrow x_u^{(l)} + \text{Mamba}\left[\left(s_{u,i}^{(l)}\right)_{i=1}^{K}\right][K]$
9: $h_{\text{global}}^{(l)} \leftarrow x_u^{(l)} + \text{Mamba}\left[\text{GS}\left[\{x_w^{(l)} | w \in U\}\right]\right][u]$
10: $h^{(l)} \leftarrow h_{\text{local}}^{(l)} + h_{\text{global}}^{(l)}$
11: $x_u^{(l+1)} \leftarrow h^{(l)} + \text{MLP}\left(h^{(l)}\right)$
12:
13: **end for**
14: *Note: [K] denotes indexing by element K, and [u] denotes indexing by the element of the sequence that coresponds to the input node* $x_u^{(l)}$.

4.4 Efficient GRED Serialization

The GRED method is difficult to implement efficiently because computing the layer requires aggregating all the the K-hop neighbours into a sequence *for each node in the graph* (line 4 in 1). In order to do so, we precomputed distance masks $\mathcal{M}$ of shape $(K, |U|, |U|)$, where $|U|$ is the number of nodes in the graph and K is the maximum-hop parameter of our model. We define $\mathcal{M}[k, u, w] = 1$ if the shortest distance between u and w is k. Given such distance masks, the serialization can be easily done my multiplying the node embeddings with such masks. Since the distance masks do not depend on the parameters of the network, they can be computed once at the beginning of training, for example with the Floyd–Warshall algorithm, and reused. Nevertheless, attempting to compute such masks in their dense form naively, as we did in the midway report, was not efficient. For example, in Peptides-struct (one of the datasets of our experiments), the average $|U|$ was around 300, K was up to 25, and we used 2500 training samples, yielding an array of size 300x300x25x2500, which is very heavy to save and load. That is why we worked with a sparse representation of the distance masks. Instead of computing $\mathcal{M}$, we computed $\hat{\mathcal{M}}$ of shape $(N, 4)$, where each of the N rows is (g, u, w, k), meaning that in the graph with identifier g, the distance between th nodes u and w was k. This sparse representation reduced training time by a factor of 8. Our code for precomputing and transforming from dense to sparse representations is available in the repository.

4.5 Oversmoothing Analysis

We use the Dirichlet Energy metric to quantitatively analyze oversmoothing. More specifically, for a trained model and a given layer l, we compute:

$$E(H^l) = \frac{1}{|V|} \sum_{v \in V} \sum_{u \in N_v} \|h_u^l - h_v^l\|^2$$

where for each node $v \in V$, we take all its neighbors $u \in N_v$ and we average the distance between these embeddings. If oversmoothing occurs, this energy would decrease when l increases, meaning that for further layers near the final one, all the embeddings are the same and the nodes which influence their neighbor's representation all have the same latent representation.

5 Results

5.1 Datasets

We implemented the three models mentioned in 4.2 and performed experiments on two datasets. The first one is the ZINC dataset (Gómez-Bombarelli et al. [2016]), which comprises 250,000 molecular graphs, for which we used a subset of 12k for training and 2k for testing. The task is a regression of a single value for the whole graph, the constrained solubility. We already reported some preliminary results of this dataset in the midway report, but now we performed new experiments with the new models and a larger training time. The other dataset is Peptides-struct (Dwivedi et al. [2023]), which comprises around 15k graphs with over 2 million nodes. We used a smaller subset of 2.5k for training and 250 for testing. The graphs represent aminoacids and the task is to regress 11 properties of such aminoacids. This dataset is known to be challenging for its long range interactions.

5.2 Baseline results

The results of our models on ZINC and Peptides-struct are shown in Table 1. On ZINC, all the models performed similarly, although GREDMamba-Mamba is slightly better than the rest. As we mentioned, this dataset consists of small graphs (the maximum number of nodes is 35), and it is not known to have very large interactions. Thus, a single message passing neural network is enough to capture all the relationships between the nodes, which is why we believe GINEConv performs as well as our models.

However, the situation changes in the Peptides-struct dataset. In this dataset, which is known to have strong long range interactions between nodes, GREDMamba is significantly better than GINEConv. This is what we expected, since Mamba is specifically designed to deal with long range interactions. Nevertheless, GREDMamba-Mamba does not perform well on this dataset. We believe this is due to the fact that the Global Sequence Model stage is not well suited to complex graphs with long rage interactions. We give an analysis on why this might be in section 6.

Dataset	Model	Test MAE	Params (k)	Time/Epoch (s)
ZINC	GINEConv (Baseline)	0.356	401	1.05
	GREDMamba (Ours)	0.370	642	5.09
	GREDMamba-Mamba (Ours)	**0.340**	881	7.80
Peptides-struct	GINEConv (Baseline)	11.42	202	2.44
	GREDMamba (Ours)	**6.17**	171	40.57
	GREDMamba-Mamba (Ours)	18.00	881	17.81

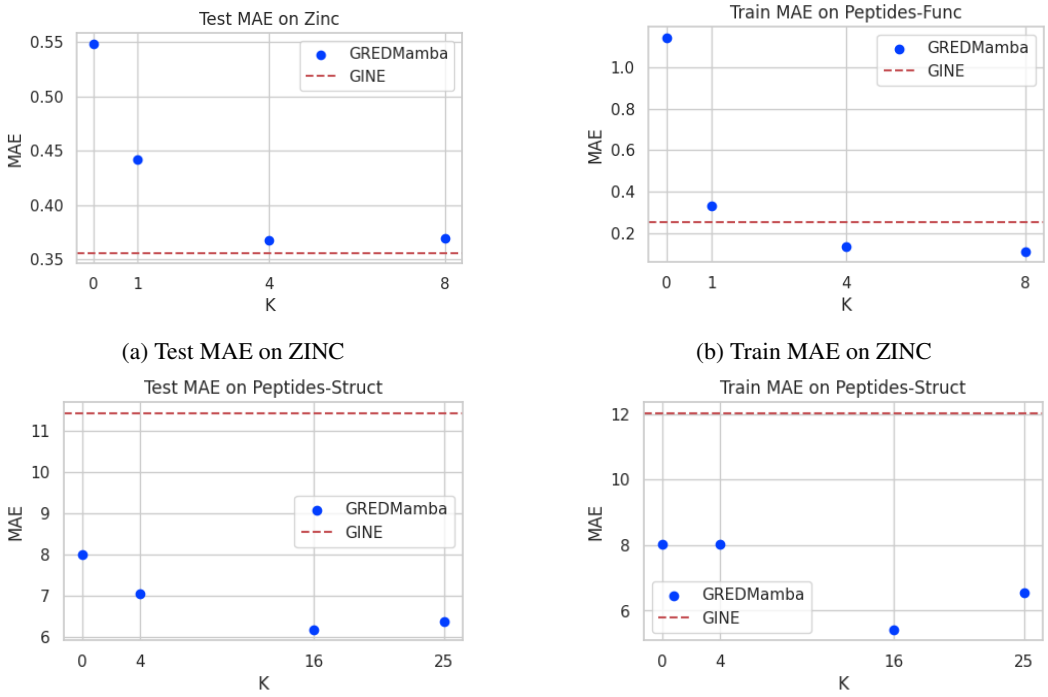
Table 1: Best Test MAE (Mean Average Error) for Zinc and Peptides-struct and each of our models, as well as number of parameters and Training Time per epoch

5.3 Ablation study: effect of K

We also study the effect of the parameter K, the furthest away distance that each node sees in every iteration of the GRED Message Passing algorithm. Having $K = 0$ would correspond to no Message Passing in the GRED layer, i.e. each node is updated based only on its value. In contrast, a very large K would mean that very far away neighbours influence the updates of each node. In figure 3, we see that the performance normally increases with K, up to a point that it saturates. In the ZINC layer, the optimal K is around 4, after that the performance does not increase when adding more Ks. In contrast, in the Peptides-struct datasaet, the optimal K is around 16, four times greater. This is again due to the differences between the datasets: ZINC is made of small graphs with mainly local interactions, whereas Peptides-struct consists of big graphs with long-range interactions.

Interestingly, we can see that GREDMamba outperforms GINEConv in the Peptides-struct dataset even when K is 0. After a careful analysis of the Peptides-struct dataset, we saw that each node is normally only connected to one or two other nodes. That makes message passing very inefficient, since after 4 layers each node can only see roughly four other nodes of the graph. Since the interactions in this graph are long range, this doesn't help much in performance, and it even seems to worsen

training and gradient propagation by forcing the architecture to model short-range interactions that do not appear to exist. In contrast, GREDMamba with $K = 0$ just ignores these short-range interactions (the same can happen with $K = 1$ if Mamba forgets past information), and tries to learn node embeddings that are useful after the global add pooling layer.



(a) Test MAE on ZINC

(b) Train MAE on ZINC

(c) Test MAE on Peptides-Struct

(d) Train MAE on Peptides-Struct

Figure 3: Effect of K (max K-hop neighbourhood for GRED, see 4.1), on train and test MAE.

5.4 Oversmoothing

In order to have a greater understanding of what the models are doing, we measured the Dirichlet energy of the GREDMamba and GINEConv models, pretrained on ZINC and evaluated also on the graphs of the ZINC dataset. The results are shown in figure 4. GREDMamba HAS a higher Dirichlet energy compared to GINE, which can be explained by the fact that GREDMamba local interactions message passing allows a more rich mixing of the nodes because it is modelling the joint probability distribution of the K-hop neighbourhood (as opposed to GINEConv, which only models the distribution of the 1-hop neighbourhood). Nevertheless, at the last layer the Dirichlet energy decreases a lot, as it is often observed empirically (note the logarithmic scale). Moreover, having a higher Dirichlet energy in this case doesn't translate into a better performance, but is is an indication of what the graph representations that the model is learning. We hypothesize that if we were to take the learnt node features of these models and used them for other downstream tasks, GREDMamba would perform better.

5.5 Training details

For both datasets, we used MSE loss as a training objective, and MAE (Mean Average Error) as evaluation metric, as it is standard in the literature. A dropout of 0.2 is used right after the Message Passing Neural Network, and in GREDMamba-Mamba it is used again right after the Global Sequence transformation. The models are trained for 200 epochs, with weight decay of 0.1, a learning rate of 0.001 and cosine scheduling. All models have used 4 layers.

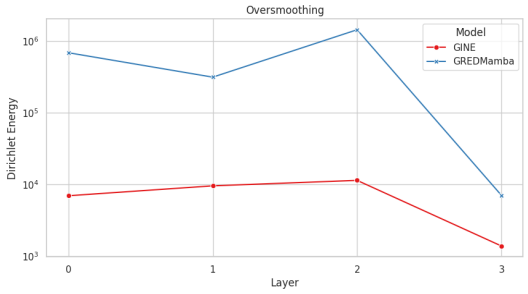


Figure 4: Differences in Dirichlet energy for the two best performing models in the ZINC dataset.

6 Discussion and Analysis

Global Sequence Model: First, we analyse why GREDMamba-Mamba performs reasonably well in ZINC but not in Peptides-struct. In the Global Sequence Model stage, the structure of the graph is lost, and the nodes are ordered based on their degree. When the number of nodes to order is small, this heuristic can work relatively well, since there are not that many ways to order them and a heuristically chosen one can be close to the optimal. Nevertheless, the number of possible orderings increases exponentially with the size of the graph. Thus, because peptides-func consists of large graphs (around 300 nodes, compared to the roughly 30 of ZINC), the heuristic order can be very far away from the optimal one. Thus, we conclude that the Global Sequence Model stage is more suited to small graphs, and harder to train on large graphs.

Overfitting: Another big challenge that we faced was overfitting. Despite using a dropout of 0.2 during training, ours models overfitted a lot in the ZINC dataset, as observed in 3b. Indeed, we can see that with $K = 8$ the performance of GREDMamba is much better than GINEConv in ZINC. However it was not easy to get rid of this overfitting, standard deep learning tools such as increasing the learning rate, changing the scheduling so that it does not decrease as much, increasing the weight decay or decreasing the number of parameters did not work. In contrast, on Peptides-Struct overfitting did not occur, which is a clear example of the big difference between graph tasks and the difficulty of training models in each of them.

Architecture Design: Related to this idea of variety in graph tasks, it is hard to design models that can be applied to all graphs without modifications. For example, GREDMamba-Mamba works relatively well on ZINC, but the exact same model performs poorly on Peptides-Struct. Despite commonly classified as graphs datasets, we argue that graphs are inherently a very flexibly data representation structure, and thus it is wrong to have a one-size-fits-all approach. Unfortunately running experiments on Peptides-Struct was computationally expensive (around 3 hours for each run in a NVIDIA RTX 6000 48 GB GPU), so we could not find a fix for GREDMamba-Mamba. Nevertheless, based on the huge importance many training and architecture details had on the performance, we believe that with more computational resources as configuration of GREDMamba-Mamba could perform at least as well as the other models.

7 Teamwork

For the final product, all team members contributed to various parts. Regarding implementations, Ricardo provided the training pipeline for the repository and implemented over-smoothing. Richa implemented GRED, graph processing tasks, and initial versions of over-smoothing and over-squashing. Ricardo ran all the experiments and actively participated in debugging and visualizing the results. Bao ran initial experiments for over-smoothing. For the final paper, Bao mostly wrote most of the Introduction, Related Work and Methods part. Ricardo wrote the Methods, Results, and Discussion and Analysis.

8 Access to Code

Our code is publicly available at https://github.com/r-buitrago/SeqForGraphs.

References

Ralph Abboud, Radoslav Dimitrov, and İsmail İlkan Ceylan. Shortest path networks for graph property prediction, 2023.

Ali Behrouz and Farnoosh Hashemi. Graph mamba: Towards learning on graphs with state space models, 2024.

Xavier Bresson and Thomas Laurent. Residual gated graph convnets, 2018.

Chen Cai and Yusu Wang. A note on over-smoothing for graph neural networks, 2020.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with performers, 2022.

Yuhui Ding. GRED: Recurrent Distance-Encoding neural networks for graph representation learning. URL `https://github.com/skeletondyh/GRED`.

Yuhui Ding, Antonio Orvieto, Bobby He, and Thomas Hofmann. Recurrent distance filtering for graph representation learning, 2024.

Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs, 2021.

Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Graph neural networks with learnable structural and positional representations, 2022.

Vijay Prakash Dwivedi, Ladislav Rampášek, Mikhail Galkin, Ali Parviz, Guy Wolf, Anh Tuan Luu, and Dominique Beaini. Long range graph benchmark, 2023.

Federico Errica, Henrik Christiansen, Viktor Zaverkin, Takashi Maruyama, Mathias Niepert, and Francesco Alesiani. Adaptive message passing: A general framework to mitigate oversmoothing, oversquashing, and underreaching, 2024.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry, 2017.

Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules, October 2016. URL `http://arxiv.org/abs/1610.02415`.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces, 2023.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces, 2022.

Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data, 2015.

Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks, 2020.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. Resurrecting recurrent neural networks for long sequences, 2023.

Ladislav Rampášek, Mikhail Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer, 2023.

T. Konstantin Rusch, Michael M. Bronstein, and Siddhartha Mishra. A survey on oversmoothing in graph neural networks, 2023.

Hamed Shirzad, Ameya Velingker, Balaji Venkatachalam, Danica J. Sutherland, and Ali Kemal Sinop. Exphormer: Sparse transformers for graphs, 2023.

Yunchong Song, Siyuan Huang, Jiacheng Cai, Xinbing Wang, Chenghu Zhou, and Zhouhan Lin. S4g: Breaking the bottleneck on graphs with structured state spaces, 2024. URL `https://openreview.net/forum?id=0Z6lN4GYrO`.

Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M. Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature, 2022.

Chloe Wang, Oleksii Tsepa, Jun Ma, and Bo Wang. Graph-mamba: Towards long-range graph sequence modeling with selective state spaces, 2024.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks?, 2019.